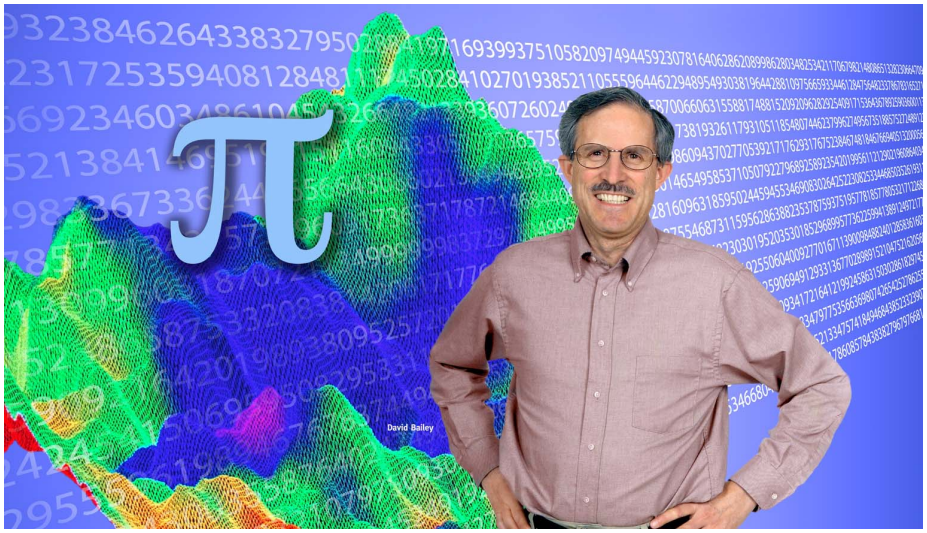
The PSLQ Algorithm: Techniques for Efficient Computation

David H Bailey

Lawrence Berkeley National Laboratory

http://crd.lbl.gov/~dhbailey



LBNL's high-precision software

- QD: double-double (31 digits) and quad-double (62 digits).
- ARPREC: arbitrary precision.
- Low-level routines written in C++.
- C++ and Fortran-90 translation modules permit use with existing C++ and Fortran-90 programs -- only minor code changes are required.
- Includes many common functions: sqrt, cos, exp, gamma, etc.
- PSLQ, root finding, numerical integration.

Available at: **http://www.experimentalmath.info**

Authors: Xiaoye Li, Yozo Hida, Brandon Thompson and DHB.

Currently being maintained with the able assistance of Alex Kaiser of LBNL.

The PSLQ integer relation algorithm

Let (x_n) be a given vector of real numbers. An integer relation algorithm finds integers (a_n) such that

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = 0$$

(or within "epsilon" of zero, where epsilon = 10^{-p} and p is the precision).

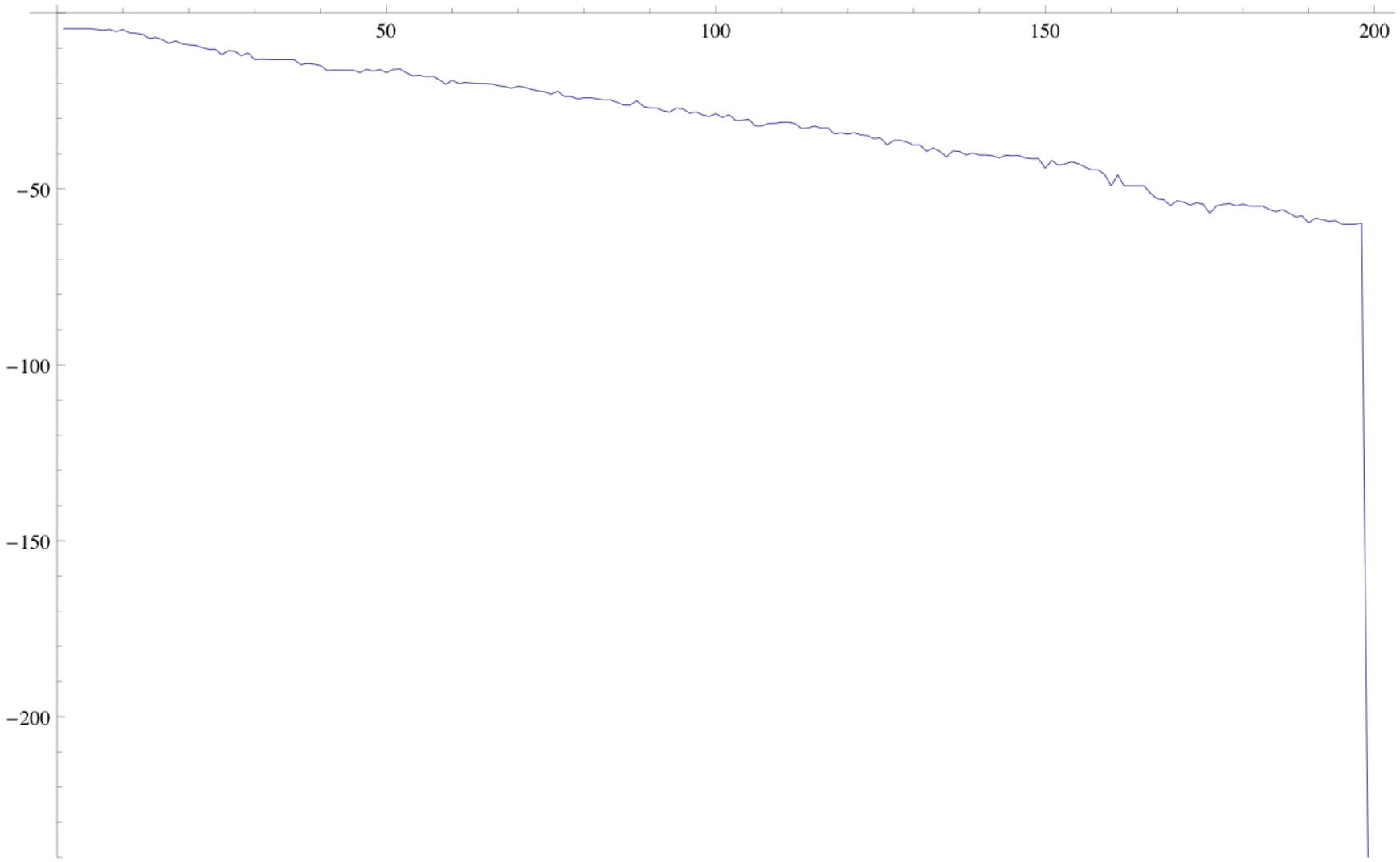
At the present time the "PSLQ" algorithm of mathematician-sculptor Helaman Ferguson is the most widely used integer relation algorithm. It was named one of ten "algorithms of the century" by *Computing in Science and Engineering*.

1. H. R. P. Ferguson, DHB and S. Arno, "Analysis of PSLQ, an integer relation finding algorithm," *Mathematics of Computation*, vol. 68, no. 225 (Jan 1999), pg. 351-369.
2. DHB and D. J. Broadhurst, "Parallel integer relation detection: Techniques and applications," *Mathematics of Computation*, vol. 70, no. 236 (Oct 2000), pg. 1719-1736.

PSLQ, continued

- PSLQ constructs a sequence of integer-valued matrices B_n that reduces the vector $y = x\,B_n$, until either the relation is found (as one of the columns of B_n), or else precision is exhausted.
- At the same time, PSLQ generates a steadily growing bound on the size of any possible relation.
- When a relation is found, the size of smallest entry of the vector y abruptly drops to roughly "epsilon" (i.e. 10^{-p}, where p is the number of digits of precision).
- The size of this drop can be viewed as a "confidence level" that the relation is real and not merely a numerical artifact -- a drop of 20+ orders of magnitude almost always indicates a real relation.
- PSLQ (or any other integer relation scheme) requires *very high precision arithmetic* (at least nd digits, where d is the size in digits of the largest a_k), both in the input data and in the operation of the algorithm.

Decrease of $\log_{10}(\min_k |y_k|)$ as a function of iteration number in a typical PSLQ run



Methodology for using PSLQ to recognize an unknown constant α

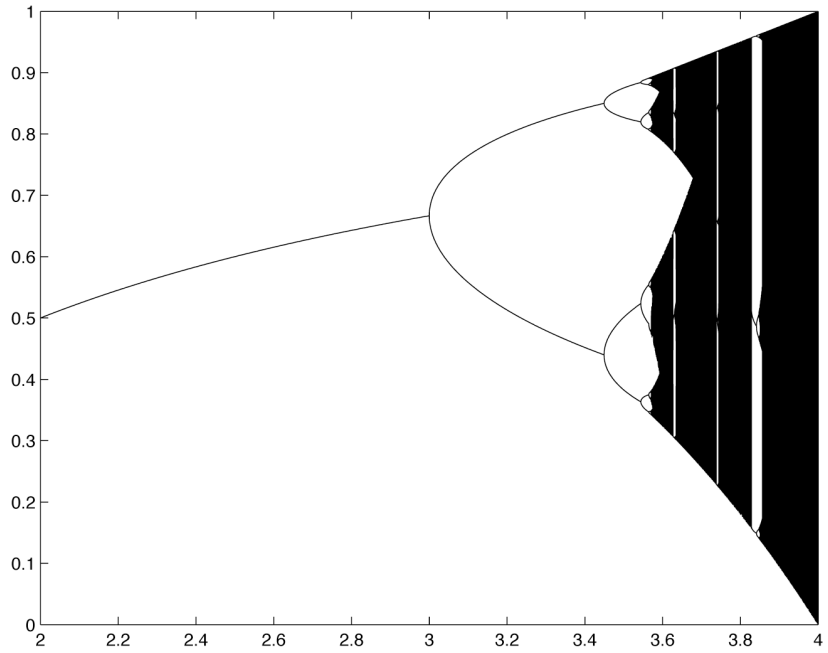
- Calculate α to high precision – typically 100 - 1000 digits. This is often the most computationally expensive part of the entire process.
- Based on experience with similar constants or relations, make a list of possible terms on the right-hand side (RHS) of a linear formula for α, then calculate each of the *n* RHS terms to the same precision as α.
- If you suspect α is algebraic of degree *n* (the root of a degree-*n* polynomial with integer coefficients), compute the vector $(1, \alpha, \alpha^2, \alpha^3, \ldots, \alpha^n)$.
- Apply PSLQ to the (*n*+1)-long vector, using the same numeric precision as α, but with a detection threshold a few orders of magnitude larger than "epsilon"– e.g., 10^{-480} instead of 10^{-500} for 500-digit arithmetic.
- When PSLQ runs, look for a detection following a drop in the size of the reduced *y* vector by at least 20 orders of magnitude, to value near epsilon.
- If no credible relation is found, try expanding the list of RHS terms.
- Another possibility is to search for multiplicative relations (i.e., monomial expressions), which can be done by taking logarithms of α and constants.

Bifurcation points in chaos theory: The first "real" application of PSLQ

Let $t = B_3$ = the smallest r such that the "logistic iteration"

$$x_{n+1} = r x_n (1 - x_n)$$

exhibits 8-way periodicity instead of 4-way periodicity.



By means of a sequential approximation scheme, one can obtain the numerical value of t to any desired precision:

3.544090359551922853615965986604804540583099845444573675457812
530305842942858863012256258566424891799962 6…

Applying PSLQ to (1, t, t^2, t^3, …, t^{12}), we obtained the result that t is a root of:

$$0 = 4913 + 2108t^2 - 604t^3 - 977t^4 + 8t^5 + 44t^6 + 392t^7 - 193t^8 - 40t^9 + 48t^{10} - 12t^{11} + t^{12}$$

Ising integrals

We recently applied our methods to study three classes of integrals that arise in the Ising theory of mathematical physics – D_n and two others:

$$C_n := \frac{4}{n!}\int_0^\infty \cdots \int_0^\infty \frac{1}{\left(\sum_{j=1}^n (u_j + 1/u_j)\right)^2} \frac{\mathrm{d}u_1}{u_1} \cdots \frac{\mathrm{d}u_n}{u_n}$$

$$D_n := \frac{4}{n!}\int_0^\infty \cdots \int_0^\infty \frac{\prod_{i<j} \left(\frac{u_i - u_j}{u_i + u_j}\right)^2}{\left(\sum_{j=1}^n (u_j + 1/u_j)\right)^2} \frac{\mathrm{d}u_1}{u_1} \cdots \frac{\mathrm{d}u_n}{u_n}$$

$$E_n = 2\int_0^1 \cdots \int_0^1 \left(\prod_{1\le j<k\le n} \frac{u_k - u_j}{u_k + u_j}\right)^2 \mathrm{d}t_2\, dt_3 \cdots \mathrm{d}t_n$$

where in the last line $u_k = t_1\, t_2 \ldots t_k$.

DHB, J. M. Borwein and R. E. Crandall, "Integrals of the Ising class," *Journal of Physics A: Mathematical and General*, vol. 39 (2006), pg. 12271-12302.

Ising integral evaluations

$$
\begin{aligned}
D_2 &= 1/3 \\
D_3 &= 8 + 4\pi^2/3 - 27\,\mathrm{L}_{-3}(2) \\
D_4 &= 4\pi^2/9 - 1/6 - 7\zeta(3)/2 \\
E_2 &= 6 - 8\log 2 \\
E_3 &= 10 - 2\pi^2 - 8\log 2 + 32\log^2 2 \\
E_4 &= 22 - 82\zeta(3) - 24\log 2 + 176\log^2 2 - 256(\log^3 2)/3 \\
&\quad +16\pi^2\log 2 - 22\pi^2/3 \\
E_5 &\stackrel{?}{=} 42 - 1984\,\mathrm{Li}_4(1/2) + 189\pi^4/10 - 74\zeta(3) - 1272\zeta(3)\log 2 \\
&\quad +40\pi^2\log^2 2 - 62\pi^2/3 + 40(\pi^2\log 2)/3 + 88\log^4 2 \\
&\quad +464\log^2 2 - 40\log 2
\end{aligned}
$$

where $\mathrm{Li}_n(x)$ is the polylog function. D_2, D_3 and D_4 were originally provided to us by mathematical physicist Craig Tracy, who hoped that our tools could help identify D_5.

Recursions in Ising integrals

Consider the 2-parameter class of Ising integrals (which arises in QFT for odd *k*):

$$C_{n,k} = \frac{4}{n!}\int_0^\infty \cdots \int_0^\infty \frac{1}{\left(\sum_{j=1}^n (u_j + 1/u_j)\right)^{k+1}} \frac{\mathrm{d}u_1}{u_1} \cdots \frac{\mathrm{d}u_n}{u_n}$$

After computing 1000-digit numerical values for all *n* up to 36 and all *k* up to 75 (performed on a highly parallel computer system), we discovered (using PSLQ) linear relations in the rows of this array. For example, when *n* = 3:

$$\begin{aligned}
0 &= C_{3,0} - 84C_{3,2} + 216C_{3,4} \\
0 &= 2C_{3,1} - 69C_{3,3} + 135C_{3,5} \\
0 &= C_{3,2} - 24C_{3,4} + 40C_{3,6} \\
0 &= 32C_{3,3} - 630C_{3,5} + 945C_{3,7} \\
0 &= 125C_{3,4} - 2172C_{3,6} + 3024C_{3,8}
\end{aligned}$$

Similar, but more complicated, recursions have been found for all *n*.

DHB, D. Borwein, J.M. Borwein and R.E. Crandall, "Hypergeometric forms for Ising-class integrals," *Experimental Mathematics*, vol. 16 (2007), pg. 257-276.

J. M. Borwein and B. Salvy, "A proof of a recursion for Bessel moments," *Experimental Mathematics*, vol. 17 (2008), pg. 223-230.

Recent result (18 Jan 2009)

$$\Delta_3(-1) = \frac{2}{\sqrt{\pi}} \int_0^\infty \frac{(-1 + e^{-u^2} + \sqrt{\pi}\, u\, \mathrm{erf}(u))^3}{u^6}\, \mathrm{d}u$$

$$= \frac{1}{15}\left(6 + 6\sqrt{2} - 12\sqrt{3} - 10\pi + 30\log(1+\sqrt{2}) + 30\log(2+\sqrt{3})\right)$$

As in many of the previous results, this was found by first computing the integral to high precision (250 to 1000 digits), conjecturing possible terms on the right-hand side, then applying PSLQ to look for a relation. We now have proven this result.

Dozens of similar results have since been found (see next few viewgraphs), raising hope that all box integrals eventually will be evaluated in closed form.

DHB, J. M. Borwein and R. E. Crandall, "Advances in the theory of box integrals," manuscript, Mar 2009, available at http://crd.lbl.gov/~dhbailey/dhbpapers/BoxII.pdf.

Recent evaluations of box integrals

n	s	$B_n(s)$
any	even $s \geq 0$	rational, e.g., : $B_2(2) = 2/3$
1	$s \neq -1$	$\frac{1}{s+1}$
2	-4	$-\frac{1}{4} - \frac{\pi}{8}$
2	-3	$-\sqrt{2}$
2	-1	$2\log(1+\sqrt{2})$
2	1	$\frac{1}{3}\sqrt{2} + \frac{1}{3}\log(1+\sqrt{2})$
2	3	$\frac{7}{5}\sqrt{2} + \frac{3}{20}\log(1+\sqrt{2})$
2	$s \neq -2$	$\frac{2}{2+s}\ {}_2F_1\left(\frac{1}{2}, -\frac{s}{2}; \frac{3}{2}; -1\right)$
3	-5	$-\frac{1}{6}\sqrt{3} - \frac{1}{12}\pi$
3	-4	$-\frac{3}{2}\sqrt{2}\arctan\frac{1}{\sqrt{2}}$
3	-2	$-3\mathrm{G} + \frac{3}{2}\pi\log(1+\sqrt{2}) + 3\ \mathrm{Ti}_2(3-2\sqrt{2})$
3	-1	$-\frac{1}{4}\pi + \frac{3}{2}\log\left(2+\sqrt{3}\right)$
3	1	$\frac{1}{4}\sqrt{3} - \frac{1}{24}\pi + \frac{1}{2}\log\left(2+\sqrt{3}\right)$
3	3	$\frac{2}{5}\sqrt{3} - \frac{1}{60}\pi - \frac{7}{20}\log\left(2+\sqrt{3}\right)$

Here *F* is hypergeometric function; *G* is Catalan; Ti is Lewin's inverse-tan function.

Elliptic Integrals

Recent research in integrals of elliptic functions have revealed hundreds of heretofore unknown identities, for instance:

$$\begin{aligned}
-2\int_0^1 xK(x)\,\mathrm{d}x + 3\int_0^1 xE(x)\,\mathrm{d}x &= 0 \\
2\int_0^1 K^2(x)\,\mathrm{d}x - 4\int_0^1 K(x)E(x)\,\mathrm{d}x + 3\int_0^1 E^2(x)\,\mathrm{d}x - \int_0^1 K'(x)E'(x)\,\mathrm{d}x &= 0 \\
-2\int_0^1 K^3(x)K'(x)E'(x)\,\mathrm{d}x + \int_0^1 E(x)K'^3(x)E'(x)\,\mathrm{d}x &= 0
\end{aligned}$$

These studies involved computing thousands of individual definite integrals, each to at least 1600-digit precision, then searching for relations among them using PSLQ.

Statement of the PSLQ algorithm

Initialization:

1. For $j := 1$ to n: for $i := 1$ to n: if $i = j$ then set $A_{ij} := 1$ and $B_{ij} := 1$ else set $A_{ij} := 0$ and $B_{ij} := 0$; endfor; endfor.

2. For $k := 1$ to n: set $s_k := \sqrt{\sum_{j=k}^{n} x_j^2}$; endfor. Set $t = 1/s_1$. For $k := 1$ to n: set $y_k := tx_k$; $s_k := ts_k$; endfor.

3. Initial H: For $j := 1$ to $n-1$: for $i := 1$ to $j-1$: set $H_{ij} := 0$; endfor; set $H_{jj} := s_{j+1}/s_j$; for $i := j+1$ to n: set $H_{ij} := -y_i y_j/(s_j s_{j+1})$; endfor; endfor.

4. Reduce H: For $i := 2$ to n: for $j := i-1$ to 1 step -1: set $t := \mathrm{nint}(H_{ij}/H_{jj})$; and $y_j := y_j + ty_i$; for $k := 1$ to j: set $H_{ik} := H_{ik} - tH_{jk}$; endfor; for $k := 1$ to n: set $A_{ik} := A_{ik} - tA_{jk}$ and $B_{kj} := B_{kj} + tB_{ki}$; endfor; endfor; endfor.

Statement of the PSLQ algorithm, continued

Iteration:

1. Select m such that $\gamma^i |H_{ii}|$ is maximal when $i = m$.

2. Exchange the entries of y indexed m and $m+1$, the corresponding rows of A and H, and the corresponding columns of B.

3. Remove corner on H diagonal: If $m \leq n-2$ then set $t_0 := \sqrt{H_{mm}^2 + H_{m,m+1}^2}$, $t_1 := H_{mm}/t_0$ and $t_2 := H_{m,m+1}/t_0$; for $i := m$ to n: set $t_3 := H_{im}$, $t_4 := H_{i,m+1}$, $H_{im} := t_1 t_3 + t_2 t_4$ and $H_{i,m+1} := -t_2 t_3 + t_1 t_4$; endfor; endif.

4. Reduce H: For $i := m+1$ to n: for $j := \min(i-1, m+1)$ to 1 step -1: set $t := \mathrm{nint}(H_{ij}/H_{jj})$ and $y_j := y_j + t y_i$; for $k := 1$ to j: set $H_{ik} := H_{ik} - t H_{jk}$; endfor; for $k := 1$ to n: set $A_{ik} := A_{ik} - t A_{jk}$ and $B_{kj} := B_{kj} + t B_{ki}$; endfor; endfor; endfor.

5. Norm bound: Compute $M := 1/\max_j |H_{jj}|$. Then there can exist no relation vector whose Euclidean norm is less than M.

6. Termination test: If the largest entry of A exceeds the level of numeric precision used, then precision is exhausted. If the smallest entry of the y vector is less than the detection threshold (see below), a relation has been detected and is given in the corresponding column of B.

Multi-level implementations of PSLQ

- In spite of the effectiveness of PSLQ, computation time grows cubically with the vector size *n* and also increases sharply with the level of numerical precision employed.
- Huge savings in run time can be achieved by employing a "two-level" implementation of PSLQ: Perform most iterations with simple double-precision arithmetic, periodically updating the multiprecision arrays.
- Because double-precision arithmetic is so much faster, savings of up to 100X can be achieved.
- For very large *n* and very high precision, additional savings can be achieved by employing a "three-level" scheme: double precision, "intermediate precision" (typically 120 digits), and full multiprecision.

Details of two-level PSLQ

1. Perform standard PSLQ initialization.
2. Perform DP "re-initialization": set $\underline{A}$ and $\underline{B} = I$, $\underline{y} = y$ (scaled so max = 1.0), and $\underline{H} = H$ (here underscore denotes DP approximations).
3. Perform LQ matrix factorization of $\underline{H}$, and replace $\underline{H}$ with lower-diagonal.
4. Perform PSLQ iterations using DP arrays. When the largest entry of A and B exceeds 10^{13}, or if a very small entry is detected in the y vector, update full-precision y, A, B, H arrays by matrix multiplication.
5. Check full-precision y vector for an entry that is zero (or smaller than some acceptable epsilon). If so, a relation has been detected.
6. Compute norm bound.
7. Repeat beginning with step 2.

Notes:

- Computation of full-precision A matrix may be omitted.
- Full-precision H matrix will not be in lower-triangular form, and thus cannot be used to compute norm bounds, but $\underline{H}$ can be used for this.

Some potential difficulties

- Occasionally a very large entry ($> 2^{53} = 9 \times 10^{15}$ approx.) is produced in the double precision $\underline{A}$ or $\underline{B}$ matrix, in spite of the 10^{13} cutoff.
- One must also check if any *intermediate* value produced in the computation of any entry of $\underline{A}$ or $\underline{B}$ exceeds 2^{53}.
- When this happens, it is necessary to:
  1. Abandon the current set of DP iterations.
  2. Retreat to stored values of $\underline{y}$, $\underline{A}$, $\underline{B}$ and $\underline{H}$ at some previous iteration.
  3. Update the multiprecision arrays using these stored values.
  4. Perform a full LQ matrix factorization on H.
  5. Perform multiprecision iterations, checking periodically (typically every ten iterations) to see if the dynamic range of the y vector has reduced to the point that DP iterations are safe again.
- A three-level implementation saves time on very large problems above a two-level scheme, mostly because intermediate precision can be used to handle these special cases instead of full precision.

Performance results for 1-2-3 level PSLQ: Recover polynomial for alpha = 3^1/r – 2^1/s

			One-level		Two-level		Three-level	
r, s	n	Iterations	Digits	Time	Digits	Time	Digits	Time
5,5	26	5143	180	32.37	190	1.29		
5,6	31	9357	240	105.48	250	3.16		
6,6	37	15217	310	298.85	320	7.19		
6,7	43	25361	420	942.66	420	17.22		
7,7	50	36947	500	2363.71	510	36.29		
7,8	57	60817			680	90.08		
8,8	65	86684			850	195.19	910	233.48
8,9	73	124521			1050	425.67	1120	460.34
9,9	82	174140			1310	934.96	1370	922.90
9,10	91	245443			1620	2032.69	1680	1780.65
10,10	101	342931			2000	4968.64	2060	3366.92

The "multi-pair" PSLQ algorithm

- The standard PSLQ algorithm (even the basic one-level scheme) is poorly suited to parallel processing, because of recursions in the inner loop of the reduction step.
- In attempt to devise a scheme better suited for parallel processing, DHB developed the "multi-pair" PSLQ algorithm.
- The multi-pair PSLQ algorithm selects certain adjacent pairs of indices that can be reduced in one step, independently of the other pairs, thus providing an opportunity for parallel processing.
- As it turns out, the multi-pair PSLQ algorithm runs faster even on a single processor.
- Two-level and three-level multi-pair PSLQ schemes have been devised. Parallel versions of these have also been devised, albeit it with great difficulty.
- DHB has used "multi-pair" PSLQ (mostly 1- and 2-level) exclusively in recent research work.

Statement of the multi-pair PSLQ algorithm

Initialize:

1. For $j := 1$ to n: for $i := 1$ to n: if $i = j$ then set $A_{ij} := 1$ and $B_{ij} := 1$ else set $A_{ij} := 0$ and $B_{ij} := 0$; endfor; endfor.

2. For $k := 1$ to n: set $s_k := \sqrt{\sum_{j=k}^{n} x_j^2}$; endfor; set $t = 1/s_1$; for $k := 1$ to n: set $y_k := tx_k$; $s_k := ts_k$; endfor.

3. Initial H: For $j := 1$ to $n-1$: for $i := 1$ to $j-1$: set $H_{ij} := 0$; endfor; set $H_{jj} := s_{j+1}/s_j$; for $i := j+1$ to n: set $H_{ij} := -y_i y_j/(s_j s_{j+1})$; endfor; endfor.

Statement of the multi-pair PSLQ algorithm, continued

Iterate.

1. Sort the entries of the $(n-1)$-long vector $\{\gamma^i|H_{ii}|\}$ in decreasing order, producing the sort indices.

2. Beginning at the sort index m_1 corresponding to the largest $\gamma^i|H_{ii}|$, select pairs of indices (m_i, m_i+1), where m_i is the sort index. If at any step either m_i or m_i+1 has already been selected, pass to the next index in the list. Continue until either βn pairs have been selected, or the list is exhausted. Let p denote the number of pairs actually selected in this manner.

3. For $i := 1$ to p, exchange the entries of y indexed m_i and m_i+1, and the corresponding rows of A, B and H; endfor.

4. Remove corners on H diagonal: For $i := 1$ to p: if $m_i \le n-2$ then set $t_0 := \sqrt{H^2_{m_i,m_i} + H^2_{m_i,m_i+1}}$, $t_1 := H_{m_i,m_i}/t_0$ and $t_2 := H_{m_i,m_i+1}/t_0$; for $i := m_i$ to n: set $t_3 := H_{i,m_i}$; $t_4 := H_{i,m_i+1}$; $H_{i,m_i} := t_1 t_3 + t_2 t_4$; and $H_{i,m_i+1} := -t_2 t_3 + t_1 t_4$; endfor; endif; endfor.

Statement of the multi-pair PSLQ, continued

6. Reduce H: For $i := 2$ to n: for $j := 1$ to $n - i + 1$: set $l := i + j - 1$; for $k := j+1$ to $l-1$: set $H_{lj} := H_{lj} - T_{lk} H_{kj}$; endfor; set $T_{lj} := \text{nint}(H_{lj}/H_{jj})$ and $H_{lj} := H_{lj} - T_{lj} H_{jj}$; endfor; endfor.

7. Update y: For $j := 1$ to $n - 1$: for $i := j + 1$ to n: set $y_j := y_j + T_{ij} y_i$; endfor; endfor.

8. Update A and B: For $k := 1$ to n: for $j := 1$ to $n - 1$: for $i := j + 1$ to n: set $A_{ik} := A_{ik} - T_{ij} A_{jk}$ and $B_{jk} := B_{jk} + T_{ij} B_{ik}$; endfor; endfor; endfor.

9. Norm bound: Compute $M := 1/\max_j |H_{jj}|$. Then there can exist no relation vector whose Euclidean norm is less than M.

10. Termination test: If the largest entry of A exceeds the level of numeric precision used, then precision is exhausted. If the smallest entry of the y vector is less than the detection threshold (see section 2), a relation has been detected and is given in the corresponding row of B.

Performance results for multi-pair PSLQ: Recover polynomial for alpha = 3^1/r – 2^1/s

			One-level		Two-level		Three-level	
r, s	n	Iterations	Digits	Time	Digits	Time	Digits	Time
5,5	26	558	180	26.08	180	1.48		
5,6	31	840	230	70.71	240	3.43		
6,6	37	1136	310	189.27	310	7.84		
6,7	43	1625	400	479.07	410	17.22		
7,7	50	2071	500	1130.85	500	35.64		
7,8	57	2410			660	69.39		
8,8	65	3723			800	169.62	880	214.66
8,9	73	4943			1010	358.07	1100	427.29
9,9	82	6169			1260	744.20	1320	804.51
9,10	91	7850			1560	1556.37	1600	1450.29
10,10	101	10017			1890	3283.08	1950	2747.12

Large test problems

1. "Fibonacci":
   - Studies a conjecture of Broadhurst on the dimension of certain Euler sums.
   - $n = 145$.
   - 5,000 digits.
2. "Bifurcation":
   - Finds the polynomial satisfies by the 8-16 bifurcation in the logistic iteration.
   - $n = 120$.
   - 10,000 digits.
3. "S(20)":
   - Finds an analytic evaluation of certain constants involving Clausen values.
   - $n = 118$.
   - 5,000 digits.
4. "Ladder":
   - Finds a relation among constants associated with a root of Lehmer's poly.
   - $n = 125$.
   - 50,000 digits.

Parallel results on three test problems

Processors	Fibonacci		B_4		$S(20)$	
	Time	Speedup	Time	Speedup	Time	Speedup
1	47788	1.00	90855	1.00	23208	1.00
2	24665	1.94	46134	1.97	11973	1.94
4	12945	3.69	23966	3.79	6305	3.68
8	7076	6.75	12924	7.03	3470	6.69
16	4180	11.43	7424	12.24	2126	10.92
32	2994	15.96	4865	18.68	1548	14.99
48	2463	19.40	4049	22.44	1303	17.81

Work to be done

- These results are nearly 10 years old, employed an old all-Fortran multiprecision library and the OpenMP parallel model.
- The ARPREC package does not lend itself to OpenMP parallelization – each individual “thread” requires too much context.
- The MPI model can be used, but only for very large problems, because of greater data transfer overhead.
- The most likely parallel target for a parallel implementation is a shared-memory, multicore system (such as recent Apple workstations).
- The UPC and Co-Array Fortran are more appropriate for such an environment.

Work to be done, continued

In addition, perhaps the whole approach to high-precision computation for experimental math applications needs to be re-thought:

- One of the key developers of the QD and ARPREC libraries, is no longer interesting in doing development or maintenance.
- Recent progress by the GMP and MPFR packages show good promise – they out-performs ARPREC in certain precision levels, particularly above 1000 digits.
- The ARPREC high-level C++ and Fortran-90 translation modules continue to be very effective (high-level modules for GMP
- Can we “marry” the high-level ARPREC modules with the low-level GMP modules?

Outside the box:

- Are there other, completely novel programming environments that we should consider for this type of computation?
- Can we devise PSLQ-like algorithms for more general spaces?

Summary

- The emerging “experimental” methodology in mathematics and mathematical physics often requires hundreds or even thousands of digits of precision.
- High-precision evaluation of integrals, followed by constant-recognition techniques, has been a particularly fruitful area of recent research, with many new results in pure math and mathematical physics.
- The PSLQ algorithm continues to be extremely effective in identifying constants and in finding relationships between constants.
- Multi-level and multi-pair variants of PSLQ are much faster and more effective than standard PSLQ.
- Much work needs to be done in developing new, parallel PSLQ software, and in improving multiprecision software in general.